are unlikely to arise but they often do so when other problems are considered. There may be 20 perfectly good and equivalent ways of going through a particular process but the user may well insist on one and only one of these.

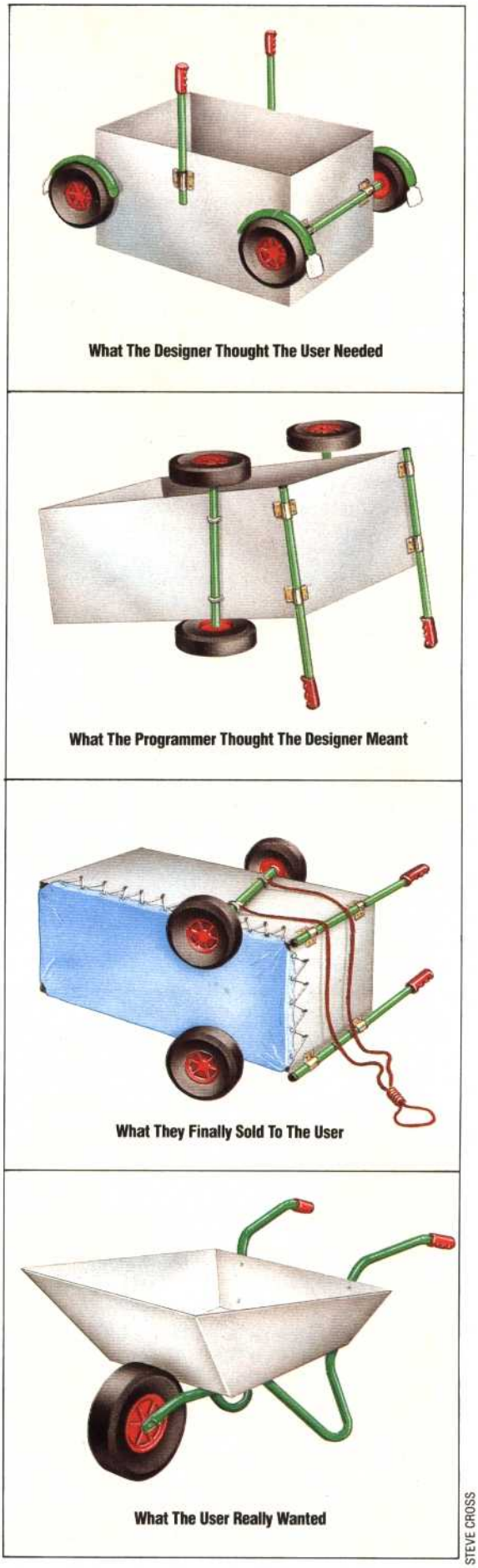
Getting this wrong makes the user immediately dissatisfied with the program. The designer may be tempted away from the user's preferred method by the greater efficiency of other methods, but any advantage is quickly lost if the user won't use the program! Discovering the user's procedures can be very helpful when it comes to designing calculations. Why invent a formula for calculating wing-loadings, for example, if you can simply ask the model maker how he does it?

With all this information noted down, the job of translating the specification into a program can begin. A useful approach to take is to design the user-program dialogue first, then the data files and then the processes that control it all. The word 'dialogue' is taken to mean the two-way communication of information that goes on between the user and the program. This does not simply consist of the input of model aircraft details and their subsequent display but also includes every prompt, message and menu that the program produces and every input, command or selection that the user enters. It is also important to fix the style of dialogue at this stage. For the aircraft program a choice between menu and command-driven interactions might be appropriate. The decisions taken here will have a considerable effect on the structure of the overall program. The contents and format of the dialogue must be considered in detail, but the reward for this effort is that all the data manipulated by the program should now be specified. This means that the storage space required for error messages and prompts can be calculated and, most importantly, the files can now be designed.

For the model aircraft program, where files will contain large blocks of text and will be very lengthy, splitting the file up onto several tapes so that each can be searched more quickly may be the best solution. If it warrants the effort, the data may be compressed by a coding algorithm before it is written to tape, and then decoded on reading.

By this time, the necessary functions will be apparent. There will be routines to allow the data text to be added and edited, to file the newly input text (these should update any indexes used by the system), to accept component names, to search for and display descriptions, etc. All these must be presented as options to the user, and they must all be able to deal with invalid data.

At this stage it is advisable for the user to make a careful check on the design to ensure that it performs as it should. If all is well, the program can now be coded. Of course, this is easier said than done, and the act of turning the design into an efficient working program may well reveal further problems.



What The Designer Thought The User Needed

What The Programmer Thought The Designer Meant

What They Finally Sold To The User

What The User Really Wanted

STEVE CROSS

Describe, Define, Design
If the typical software development team of user, designer and programmer had set out to solve the problem of moving heavy loads around gardens, this is what they might have produced. Bad communication — between expert and non-expert, and amongst experts — is still a major problem facing all design teams